Приложение

1. Реляционная модель логического разграничения доступа (RelBAC)

В настоящем Приложении кратко представлена реляционная модель логического разграничения доступа (ЛРД) к объектам информационных систем. Эта модель специально разработана для управления доступом к объектам многопользовательских систем управления сетевым контентом. При разработке реляционной модели ЛРД были приняты во внимание такие характерные особенности подобных систем, как непостоянное множество пользователей системы, большое количество разновидностей объектов и отношений между ними. Принималась во внимание также необходимость учитывать при предоставлении пользователю прав доступа к объектам опосредованные взаимосвязи между пользователем и целевым объектом. Такие взаимосвязи возникают вследствие существования объекта, связанного определенным образом как с пользователем, так и с целевым объектом. Примером такого опосредованного отношения является предоставление ответственному по подразделению прав доступа к учетным записям сотрудников этого подразделения.

Далее представлено краткое описание основных аспектов реляционной модели ЛРД. Подробное описание этой модели приведено в [1] и [2].

1.1 Определение реляционной модели логического разграничения доступа

Далее в настоящем подразделе кратко рассмотрены основные элементы реляционной модели ЛРД.

Определение 1 *Пусть в информационной системе заданы следующие множества.*

- *Objects – множество объектов системы. Для ИАС «ИСТИНА» объектами являются пользователи системы, а также зарегистрированные в системе организации и подразделения, их сотрудники (в том числе и не*

зарегистрированные как пользователи) и результаты их научно-исследовательской деятельности (например, статьи, монографии, доклады на конференциях), информация о которых хранится в системе.

- *Users⊂Objects – множество пользователей системы, входящее в множество ее объектов.*
- *Classes - множество классов объектов системы. В ИАС «ИСТИНА» входят такие классы, как «пользователь», «сотрудник», «статья» и другие.*
- *Relations=PrimitiveRelations⊔InducedRelations – множество видов отношений между объектами. Примитивными отношениями (также называемыми базовыми или прямыми) называются взаимосвязи между объектами, указанные в базе данных системы. Примерами таких отношений являются отношение места работы, связывающее сотрудника с подразделением и отношение авторства, связывающее сотрудника со статьей. Для каждого вида отношения жестко заданы классы объектов, которые могут быть соединены этим отношением. В системе также заданы порожденные отношения, для которых пары связанных объектов определяются с помощью цепочек отношений. Данное понятие описано далее в настоящем разделе.*
- *Actions – множество возможных действий над объектами. Политика безопасности определяет набор правил, согласно которому для каждой конкретной тройки (u,a,o)∈Users×Actions×Objects, разрешается, либо запрещается доступ a пользователя и к объекту o. Для некоторых классов объектов некоторые виды доступа могут не иметь смысла – например, доступ «просмотр рейтинга» возможен только для сотрудников. Неприменимость какого-либо вида доступа к определенному объекту означает его запрет для всех пользователей.*

- *Chains – множество правил, определяющих два объекта системы как связанные определенным порожденным отношением, если существует последовательность объектов, связывающая эти два объекта определенной последовательностью базовых отношений.*

- *AllowedActions⊂Relations×Actions – множество пар (r,a), означающее, что если пользователь и связан с объектом o отношением r, то он имеет право доступа a к объекту o, для всех таких видов доступа a, что (r,a)∈AllowedActions.*

Тогда будем считать, что в системе задана реляционная модель логического разграничения доступа.

Данное определение можно наглядно описать следующим образом. Система представляется в виде графа, называемого социальным графом, вершинами которого являются ее объекты. Два объекта соединены ребром, если они связаны каким-либо отношением, причем с каждым ребром ассоциировано имя соответствующего отношения. Заметим, что ребра могут быть кратными, то есть два объекта могут быть связаны несколькими разными отношениями. Доступ *a* пользователя *u* к объекту *o* разрешается в том и только том случае, если в графе существует путь от вершины *u* к вершине *o*, соответствующий одной из цепочек отношений, разрешающих доступ *a*.

Заметим, что в случае ИАС «ИСТИНА», как и других систем, разработанных на основе Django, социальный граф является нагруженным, то есть с вершинами и ребрами могут быть ассоциированы дополнительные атрибуты. Такими атрибутами являются, например, дата начала и конца периода работы сотрудника в подразделении для отношения места работы, или дата публикации для статьи. Значения атрибутов объектов могут учитываться при принятии решения о предоставлении пользователю права доступа к объекту. Для этого с некоторыми из цепочек отношений ассоциированы предикаты, зависящие от атрибутов объектов и

отношений, входящих в цепочку. Если две вершины социального графа соединены путем, соответствующим данной цепочке отношений, то они считаются соединенными порожденным цепочкой отношением только в том случае, если для вершин и ребер, входящих в этот путь соответствующий предикат истинен.

Определение 2 *Пусть – цепочка отношений, и – пользователь системы, а о – объект. Последовательность называется цепочкой объектов, соответствующей цепочке отношений с, если для любого $i \in \{1,\ldots,n\}$ объекты o_{i-1} и o_i связаны отношением r_i и при этом для атрибутов объектов выполнено условие цепочки с. Пользователь и называется связанным с объектом о цепочкой отношений с, если существует такая соответствующая с цепочка объектов (o_0, …, 0_n), что o_0=и и o_n=о.*

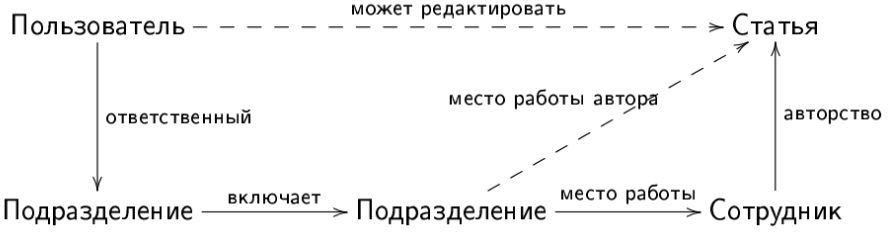

Рисунок 1: Пример цепочки отношений

Рассмотрим пример. На рисунке 1 представлен пример цепочки отношений, позволяющей ответственному по подразделению редактировать публикации сотрудников этого подразделения и его дочерних подразделений. Отношение «ответственного по статье», разрешающее доступ на редактирование, порождается цепочкой из четырех отношений: отношения ответственности пользователя по подразделению; отношения вложенности подразделений (для простоты здесь считается, что это отношение также связывает подразделения с собой с дочерними

подразделениями любого уровня вложенности – то есть является рефлексивным и транзитивным); отношения места работы сотрудника в подразделении; отношения авторства статьи. Данная цепочка имеет предикат, означающий, что она порождает отношение «ответственного по статье» только в том случае, если дата публикации статьи входит в период работы сотрудника в подразделении. В действительности с этой цепочкой связан еще один предикат, не отмеченный на рисунке, а именно – право ответственного по подразделению должно быть действительным в данный момент, то есть текущая дата должна лежать между датами начала и конца действия полномочий, ассоциированными с отношением ответственного по подразделению. Если для пользователя *u* и статьи *a* существуют такие подразделения d_1, d_2 и сотрудник *w*, что цепочка объектов (u, d_1, d_2, w, a) соответствует представленной на рисунке цепочке отношений, то пользователь *u* считается ответственным по статье *a* и имеет право редактировать ее.

Следует заметить, что среди объектов системы есть объекты, определяющие связи между другими объектами – например, объект связки подразделения с ответственным пользователем, а также объект, означающий связь между статьей и ее автором (в дальнейшем называемый объектом авторства статьи), или между сотрудником и подразделением, в котором он работает (в дальнейшем – объект места работы). При построении модели системы такие объекты могут рассматриваться как объекты, но также могут и рассматриваться как отношения. Каждый из вариантов имеет свои достоинства и недостатки. Например, использование отношения места работы сотрудника в подразделении в качестве отдельного объекта требует внесения в систему дополнительного класса объекта «место работы» и двух новых отношений, одно из которых связывает место работы с сотрудником, к которому оно относится, а другое – с подразделением, к которому это место работы относится. Отношение места работы, связывающее сотрудника с подразделением, в котором он работает при этом становится порожденным цепочкой, состоящей из двух указанных базовых отношений. Таким образом,

выделение места работы сотрудника в подразделении в отдельный объект требует включения в модель одного дополнительного класса объектов, двух дополнительных отношений и одной дополнительной цепочки отношений. Вместе с тем, использование места работы в качестве объекта позволяет регламентировать доступ к нему отдельно от доступа к связанному с ним сотруднику, либо к подразделению.

Для некоторых видов «связующих объектов» выделение в отдельный класс необходимо по причине того, что эти объекты могут использоваться, даже не имея одной из полагающимся им «связей». Например, объект «авторство статьи» содержит ссылки на статью, к которой относится, и на сотрудника, являющегося ее автором. Однако, ссылка на сотрудника может быть пустой и не указывать ни на какого зарегистрированного в системе сотрудника, если при добавлении статьи пользователь не смог выбрать ее автора из предложенного системой списка сотрудников, соответствующих указанному в статье имени автора. Для такой статьи пользователь, имя которого соответствует имени, указанному в записи об авторстве статьи, не ассоциированной с конкретным сотрудником, имеет право заявить о своем авторстве. Формулировка этого правила для разрешения доступа «заявить о своем авторстве» требует, чтобы запись об авторстве статьи считалась отдельным объектом.

Таким образом, для достижения наибольшей детализации правил разграничения доступа к объектам системы, имеет смысл любое отношение между объектами вида «многие-ко-многим» считать порожденным. Для записи в таблице этого отношения создается отдельный класс объектов – такой как, например, «место работы» или «авторство статьи», для которого в системе регистрируются отношения, связанные его с объектами, на которые он ссылается – место работы, например, имеет ссылки на объекты сотрудника и подразделения. Само отношение места работы, связывающее подразделение с сотрудником, при этом является порожденным цепочкой из двух отношений, первое из которых связывает

подразделение со всеми записями о месте работы сотрудников в этом подразделении, а второе – место работы с соответствующим ему сотрудником.

1.2 Соответствие терминов, принятых при описании реляционной модели логического разграничения доступа и библиотеки Django

При использовании реляционной модели ЛРД в системах, разработанных на основе Django, термины, принятые при описании реляционной модели, соответствуют терминам, принятым при использовании библиотеки следующим образом.

- **Класс** реляционной модели ЛРД является **моделью** Django. При этом модель является классом в терминах языка Python. С точки зрения СУБД каждая модель соответствует таблице в базе данных.
- **Объект** реляционной модели является **записью** в соответствующей модели Django. Он также является объкетом соответствующего класса языка Python. С точки зрения СУБД каждая запись является строкой в таблице, соответствующей его модели.
- **Атрибутами** объектов некоторого класса являются **поля** данных в соответствующией модели. Исключением являются поля типа `ForeignKey` и `ManyToManyField`, которые, с точки зрения реляционной модели ЛРД являются **примитивными отношениями**. Заметим, что каждое поле `ForeignKey` и `ManyToManyField` может задавать два взаимно обратных друг к другу отношения, имя одного из которых совпадает с именем поля, а имя другого задается параметром `related_name` при определении поля.

1.3 Программные механизмы реляционной модели логического разграничения доступа

Кратко рассмотрим основные программные механизмы, которые используются для внедрения реляционной модели ЛРД в Web-приложения на языке Python. С этой целью разработано приложение, предназначенное для автоматического создания подпрограммы проверки прав доступа пользователя к объекту на основе описания реляционной модели ЛРД в XML-подобном формате. Такое программное средство, называемый компилятором реляционной модели ЛРД, считывает из входного файла описание модели ЛРД, реализующей политику безопасности подконтрольной системы, которое включает определения используемых классов, отношений и описания цепочек отношений. На основе описания создается несколько функций на языке Python, одна из которых проверяет право конкретного пользователя осуществлять конкретную операцию над определенным целевым объектом, а другая – возвращает список всех операций, которые имеет право выполнять этот пользователь над определенным целевым объектом. Алгоритмы работы этих функций выглядят аналогичным образом. Функция, проверяющая право пользователя на совершение одиночной операции над объектом, последовательно выполняет запросы к базе данных, которые, в свою очередь, проверяют существование цепочек объектов, связывающих пользователя с целевым объектом каждой из допустимых цепочек отношений. В случае, если для одной из цепочек отношений, разрешающих пользователю звпрашиваемый им вид доступа к объекту, существует соответствующая цепочка объектов, связывающая пользователя с целевым объектом, функция возвращает значение True, означающее, что доступ разрешается. В случае, если ни одна их таких цепочек не соединяет пользователя с целевым объектом, возвращается значение False, означающее, что доступ запрещается.

Функция, возвращающая список всех операций, разрешенных определенному пользователю для заданного целевого объекта, работает аналогичным образом. Для каждой из цепочек отношений, которые могут соединять пользователя с объектами того же класса, что и целевой объект, выполняется запрос к базе данных, в ответ на который проверяется существование цепочки объектов, связывающей пользователя с целевым объектом этой цепочкой отношений. В случае, если такая цепочка объектов существует, в список разрешенных видов доступа добавляются виды доступа, разрешенные для отношений, порожденных этой цепочкой. В случае, если в системе применяются запрещающие правила, также ведется общий список запрещенных видов доступа, в который добавляются виды доступа, запрещенные отношениями, порожденными данной цепочкой. После проверки всех цепочек отношений функция возвращает разность между итоговыми множествами разрешенных и запрещенных видов доступа.

Отметим, что функции, созданные компилятором реляционной модели ЛРД, не содержат циклов и условных переходов, и поэтому выполняются быстро и не могут зацикливаться. Эти функции выполняют фиксированную последовательность запросов к базе данных системы, в каждом из которых изменяются только значения идентификаторов пользователя и целевого объекта. В зависимости от того, пуст или не пуст результат каждого из запросов к базе данных, модифицируется возвращаемое значения. Эти функции, таким образом, обращаются к базе данных системы в обход моделей Django, что увеличивает производительность их реализации за счет отсутствия затрат времени на вызов методов «промежуточного слоя» Django. Вместе с тем, для создания таких функций требуется, чтобы описание модели ЛРД системы включало в себя имена всех используемых таблиц и полей базы данных системы. Необходимость в таких данных вызвана тем обстоятельством, что функция проверки прав доступа не получает информации о структуре базы данных из моделей Django. Таким образом информация о структуре базы данных оказывается дублированной. Имена таблиц и

полей базы данных систему должны быть указаны как в описании модели данных в исходном коде системы, так и в описании модели ЛРД.

Создание описанного выше модуля проверки прав доступа может потребовать значительных преобразований исходного описания модели, включая операцию, именуемую приведением множества цепочек отношений к каноническому виду. Такое преобразование заключается в том, что любая цепочка, содержащая порожденное отношение, заменяется на цепочку, в которой вхождение этого отношения заменяется на подцепочку, порождающую это отношение. Таким образом, результирующее множество цепочек системы состоит из цепочек, включающих в качестве элементов только примитивные отношения. Приведение множества цепочек отношений к указанному каноническому виду необходимо для того, чтобы для каждой из цепочек вычислить SQL-запрос, выполняющий проверку существования соответствующей этой цепочке отношений цепочки объектов. Алгоритм работы компилятора реляционной модели ЛРД описан в [1] и [2].

2 Описание реляционной модели логического разграничения доступа для приложений на основе Django

В данном разделе представлены различные поддерживаемые к настоящему времени форматы описания политики безопасности, использующей реляционную модель ЛРД. В подразделе 2.1 рассматривается исторически первый из разработанных форматов описания модели, использующий XML-подобный синтаксис. Показано, что данный формат описания реляционной модели ЛРД является неудобным для сопровождения в силу ряда причин. В подразделе 2.2 рассмотрен альтернативный вариант описания модели, использующий в качестве основы язык Python и не имеющий недостатков, присущих XML-подобному формату. В подразделе 2.3 представлен ряд соображений, позволяющих сделать

описание модели разграничения доступа к объектам системы максимально удобным и интегрировать его с исходным текстом целевой системы, сохранив при этом такие преимущества использования исходного способа описания политики, как возможность ее статического анализа и верификации.

2.1 Формат входного файла компилятора реляционной модели логического разграничения доступа

В данном подразделе рассмотрен формат описания реляционной модели ЛРД, являющийся в определенном смысле базовым для всех остальных. Как отмечено в подразделе 1.3, именно этот формат ожидается на входе компилятора реляционной модели ЛРД для создания модуля проверки прав доступа. Другие форматы описания модели, представленные в 2.2 и 2.3, конвертируются в данный формат перед их передачей на вход компилятора реляционной модели ЛРД.

Базовый формат описания реляционной модели ЛРД использует XML-подобный синтаксис. Описание модели ЛРД целевой системы заключено в тег `policy`, содержимое которого разделено на следующие секции:

- список классов;
- список отношений;
- список цепочек отношений;
- список типов;
- список ролей.

Здесь ролью называется свойство пользователя, на основании которого ему разрешаются некоторые виды доступа ко всем объектам, вне зависимости от связывающих их отношений. Примерами ролей являются роль активного пользователя (неактивному пользователю запрещен вход в систему и, как следствие, все действия с объектами) и часто используемая в приложениях,

основанных на Django роль суперпользователя, которому разрешено большинство видов доступа ко всем объектам. Типом является свойство объекта определенного класса, часто используемое при проверке прав доступа к этому объекту. Использование типов является способом сокращения записи правил модели и может быть заменено включением соответствующего свойства целевого объекта в правила цепочек отношений.

В описание модели ЛРД может быть также включено описание переменных окружения, то есть данных, не содержащихся в объектах системы, но используемых при проверке прав доступа к объектам. В настоящее время единственными используемыми переменными окружения является текущая дата и текущее время.

Следует заметить, что представленная структура описания политики безопасности не соответствует структуре разделения обязанностей разработчиков ИАС «ИСТИНА», среди которых, как правило, каждый разработчик отвечает за работу с определенными классами целевых объектов. По этой причине разумным способом структурирования списка правил разграничения доступа к объектам целевой системы была бы их группировка по классам целевого объекта доступа.

2.2 Преимущества описания правил реляционной модели логического разграничения доступа на языке Python

В данном подразделе представлено описание альтернативного формата представления реляционной модели ЛРД, не имеющего недостатков, присущих формату, представленному в подразделе 2.1. С учетом того обстоятельства, что в значительной своей части исходный код ИАС «ИСТИНА» и библиотеки Django, написан на языке Python, при разработке альтернативного формата было принято решение взять за основу именно этот язык как хорошо знакомый специалистам, ответственным за сопровождение и развитие системы. Для этого был реализован отдельный программный модуль специализированной библиотеки на языке Python,

включающий функции, необходимые для формирования структуры модели ЛРД, основанной на реляционных принципах с целью ее сохранения в файл в формате, который представлен в подразделе 2.1. Таким образом, описанием модели служит сама последовательность команд, формирующая объект такой модели. Использование языка Python при этом позволяет использовать его возможности для создания сокращенных форм описания часто повторяющихся фрагментов правил разграничения доступа. В случае ИАС «ИСТИНА» примером такого часто повторяющегося фрагмента является фрагмент вида «поле начальной даты в некотором объекте меньше текущей даты, а поле конечной даты – больше текущей даты, либо не задано». При проверке прав доступа пользователей к различным объектам системы подобные условия могут использоваться для дат начала и конца срока действия полномочий ответственного по подразделению; для периода работы сотрудника в подразделении, периода членства сотрудника в диссертационном совете; для даты начала и конца сроков действия, которые ассоциированы со многими другими объектами системы.

2.3 Интеграция описания политики безопасности с описанием модели данных системы

Приложение на основе библиотеки Django включает в себя описание всех структур данных системы в виде классов, производных от `django.db.models.Model`. С учетом этого факта, одним из способов значительно упростить описание реляционной модели ЛРД является отказ от дублирования этом описании сведений, уже содержащихся в исходном коде целевой системы – например, имен таблиц и столбцов в базе данных. Для этой цели можно интегрировать библиотеку для создания структуры реляционной модели ЛРД, в код целевой системы. Библиотека Django позволяет выполнять некоторые подпрограммы Web-приложения, запуская их из командной строки без запуска Web-сервера. Таким образом, включение в код системы библиотеки для работы с реляционной моделью ЛРД позволит запускать

ее в режиме создания файла такой модели без возможности доступа к этой функции черев Web-интерфейс. Вместе с тем, библиотека для создания модели ЛРД получит возможность обращаться в моделям Django, получая из них информацию об используемой каждой моделью таблице в базе данных и ее взаимосвязях с другими моделями. Таким образом, можно избежать необходимости указывать эту информацию в описании политики безопасности системы, существенно сократив ее объем и упростив ее сопровождение.

Например, для предоставления пользователю доступа на редактирование соответствующей ему учетной записи сотрудника в примере модели сотрудника, представленном выше, может использоваться следующая команда:

```
policy.grantModelAccess (Worker,
                         nodes='worker',
                         actions=['edit'])
```

Эта команда означает, что пользователь получает право доступа edit к объекту типа Worker, с которым пользователя связывает фиктивное поле модели worker. Это поле объявлено в представленном выше примере как обратное отношение к полю user в модели Worker, которому соответствует поле user_id в таблице Workers.

Одной из трудностей, возникающих при таком подходе, является то, что имена полей внешних ключей и обратных к ним, используемые в Django, не могут однозначно идентифицировать отношение в пределах всей системы. Поле внешнего ключа с именем worker может содержаться и в других классах, содержащих ссылку на единственного пользователя – например, в классе для места работы или авторства статьи. Аналогичным образом внешний ключ с именем user есть во многих классах. По этой причине в модель введено понятие псевдонима отношения. Если имя отношения является строкой, однозначно идентифицирует отношение среди всех отношений, зарегистрированных в системе, то псевдоним

однозначно определяет отношение среди множества исходящих отношений одного класса, то есть отношений, для которых «слева» должен стоять объект определенного класса. В этом случае в качестве псевдонима отношения может использоваться имя соответствующего поля в модели Django. В случае, если имя отношения не указано при описании политики безопасности в явном виде, в качестве имени может использоваться комбинация наподобие `имя_левого_класса__псевдоним`.

Таким образом, описание реляционной модели ЛРД встраивается в целевую систему в виде последовательности команд, формирующих объект политики безопасности. Особой командой этот объект сохраняется в XML-файл, который обрабатывается компилятором реляционной модели ЛРД, который дает на выходе файл на языке Python, содержащий функции для проверки права определенного доступа пользователя к объекту и для вычисления всех видов доступа определенного пользователя к определенному объекту. Эти функции выполняются быстро и гарантированы от зацикливания в силу того обстоятельства, что каждая из них выполняет фиксированную последовательность SQL-запросов к базе данных системы, в которые подставляются только идентификаторы пользователя и целевого объекта. Созданный файл затем включается в исходный код системы, и именно эти функции выполняют проверку прав доступа к объектам в процессе работы Web-приложения. Использование такой технологии имеет следующие далее преимущества перед непосредственным выполнением команд проверки доступа к объектам, написанным на языке Python.

- Использование промежуточного, более простого по сравнению с описанием на Python, формата хранения правил разграничения доступа к объектам делает возможным статический анализ и верификацию этих правил. Например, компилятор реляционной модели ЛРД проверяет цепочки отношений на наличие зацикленных правил порождения отношений, то есть

ситуации, при которой отношение *a* порождается цепочкой, содержащей отношение *b*, а *b* – цепочкой, содержащей *a*. Выполнение алгоритма проверки прав доступа, описанного на Python, непосредственно при поступлении запроса пользователя на доступ к объекту приведет при этом к зависанию системы. Очевидно, что подобную ситуацию возможно выявить при автоматизированном тестировании системы. Вместе с тем, если зацикливание проверки прав доступа происходит в редких случаях, выявление проблемы путем модульного тестирования может потребовать настолько большого тестового покрытия, что выполнение тестов займет неприемлемое время. Вопросы построения оптимального тестового покрытия для автоматизации тестирования механизмов разграничения доступа к объектам системы рассматриваются в подразделе 3.2 настоящей статьи. Статический анализ используемой модели логического разграничения доступа к объектам системы эффективно дополняет автоматическое модульное тестирование, выявляя некоторые ошибки в описании модели, даже если вероятность того, что эта ошибка действительно произойдет, крайне мала.

- Функции проверки прав доступа созданные компилятором реляционной модели ЛРД, являются более быстрыми по сравнению с непосредственно написанными в коде целевой системы.

- Ограниченность средств, которые можно использовать при описании модели разграничения доступа к объектам системы, снижает вероятность некоторых ошибок при описании модели.

Вместе с тем, при описании модели разграничения доступа к объектам целевой системы описанным выше способом, можно использовать все механизмы языка Python для создания, например, сокращенных форм записи часто используемых видов правил. В качестве примера представлена функция, возвращающая условие,

налагаемое на промежуточный объект цепочки отношений – указанная в объекта дата начала срока его действия должна быть раньше текущей даты, а дата конца срока действия – позже текущей даты. При этом обе даты могут быть равны NULL, что означает дату в бесконечно удаленном прошлом – для даты начала срока действия объекта и дату в бесконечно удаленном будущем, как дату конца срока действия объекта.

```
def currentDateInRange (begin, end, current_date='current_date'):
    u"""
    Возвращает строку условия принадлежности текущей даты
    периоду между датами begin и end. При этом для полей
    begin и end допускается значение Null, которое для
    begin соответствует бесконечно удаленному прошлому, а
    для end - бесконечно удаленному будущему.
    :param str begin:
        Имя атрибута объекта или отношения, хранящего
        начальную дату целевого периода.
    :param str end:
        Имя атрибута объекта или отношения, хранящего
        конечную дату целевого периода.
    :param str current_date':
        Имя переменной окружения, хранящей текущую дату.
    """
    return ("((%s <= env (%s) or isNull (%s))" +
        " and (%s >= env (%s) or isNull (%s)))") \
        % (begin, current_date, begin, \
        end, current_date, end)
```

Данное правило часто используется в различных цепочках отношений для таких объектов, как запись об ответственном по подразделению, о месте работы

сотрудника и других объектов, для которых задана дата начала и конца срока их действия.

3. Автоматизация процедуры тестовых испытаний программных механизмов реляционной модели логического разграничения доступа

В настоящем разделе рассматривается решение задачи автоматизированного тестирования механизмов логического разграничения доступа, реализованных в целевой системе. Существует ряд причин, по которым в систему следует внедрить механизмы такого тестирования. Первая из этих причин состоит в том, что автоматический тест корректности работы системы можно выполнять после каждого серьезного изменения в ее исходном коде. Ручное тестирование корректности механизмов логического разграничения доступа при этом потребовало бы от разработчиков слишком много времени. Вторая причина состоит в том, что при сложной политике безопасности гарантия корректности ее соблюдения требует проверки на большом числе пользователей и объектов доступа. Это означает, что ручное тестирование механизмов безопасности системы потребует слишком большого времени. Оценка количества пользователей и целевых объектов, на примере которых необходимо проводить тестирование подсистемы логического разграничения доступа, представлена далее в подразделе 3.2 настоящей статьи. Наконец, для оценки производительности механизмов управления доступом необходимо также большое количество прогонов тестов для обеспечения статистической значимости данных о времени выполнения функции, проверяющей права доступа пользователя к объекту. Это обстоятельство также приводит к необходимости автоматизации тестовых испытаний механизмов безопасности системы.

3.1 Методы автоматизации тестирования приложений на основе Django

Библиотека Django содержит ряд инструментальных средств, которые можно использовать для тестирования различных подсистем использующего ее приложения. Такие средства имеет смысл разделить на две группы. Первую группу составляют средства тестирования приложений, входящие в стандартную библиотеку языка Python, которые были незначительно доработаны с целью их эффективного использования для тестирования модулей Web-приложений. Эта группа средств автоматизации тестирования приложений рассматривается в подразделе 3.1.1. Вторая группа средств автоматизации тестирования приложений на основе Django включает в себя тестовый Web-клиент, позволяющий эмулировать получения тестируемой системой определенных HTTP-запросов и анализировать ее ответы. Использование данного средства рассматривается в подразделе 3.1.2 настоящего Приложения.

3.1.1 Основные средства тестирования

Для тестирования поведения конкретных функций или классов системы используется модуль django.test библиотеки Django. При использовании этого модуля необходимо создать класс, производный от django.test.TestCase и включающий в себя методы, имена которых начинаются с префикса `test_`, каждый из которых выполняет тестирование поведения определенной функции в определенной ситуации. Любое приложение на основе Django можно запустить в режиме самотестирования с помощью команды:

```
python ./manage.py test
```

При этом будут выполнены все тестирующие методы во всех классах, производных от TestCase, находящихся в файлах с именами `tests.py` и других

файлах, имена которых начинаются со слова test, во всех модулях системы. При необходимости выполнить тестирование, функций, определенных в других файлах, имя файла указывается явно.

Класс django.test.TestCase является унаследованным от класса unittest.TestCase, входящего в стандартную библиотеку языка Python и описанного в [4]. В число методов этого класса входят методы, имена которых начинаются с префикса assert – например, метод assertTrue, проверяющий истинность своего аргумента и прерывающий выполнение тестового метода с сообщением об ошибке в случае, если аргумент интерпретируется как ложный. Аналогичный ему метод assertEqual принимает два аргумента и прерывает тестирование с сообщением об ошибке, если аргументы не равны. Похожим образом действуют методы assertFalse, assertNotEqual, assertAlmostEqual, assertGreater, и другием методы, описания которых доступны в документации по классам django.test.TestCase, либо unittest.TestCase. Заметим, что большинство этих методов принимают параметр `msg`, который содержит текст сообщения об ошибке, выводимого на экран при непрохождении теста. Этот параметр позволяет узнавать дополнительную полезную информацию о причинах возникшей ошибки.

Среди других методов класса django.test.TestCase следует упомянуть методы setUp и tearDown, а также метод skipTest, позволяющий отказаться от выполнения теста с указанием причины. Метод skipTest может быть вызван, например, если в базе данных систему отсутствуют объекты с необходимыми для тестирования свойствами. Метод setUp выполняется перед выполнением каждого из тестирующих методов класса и может включать подготовку данных, необходимых большинству из них. Метод tearDown выполняется после каждого тестового метода и уничтожает вспомогательные данные. Таким образом, для каждого из методов тестового класса, имя которого начинается с `test_`, выполняется следующая последовательность действий – создается экземпляр тестового класса, для него

выполняется метод setUp, затем – сам тестовый метод, после него метод tearDown, затем объект тестового класса уничтожается и для выполнения следующего тестового метода создается новый объект. По этой причине тестовый метод не должен зависеть от состояния тестового объекта после выполнения другого тестового метода – каждый из них выполняется в своем тестовом объекте.

Существует два способа обеспечить наличие в базе данных системы объектов со свойствами, обуславливающими поведение, которое необходимо протестировать. Первый из них заключается в том, чтобы создать эти объекты перед началом тестирования и удалить после его окончания. Вторая возможность заключается в том, чтобы найти объекты с нужными свойствами в постоянно работающей базе данных, предназначенной специально для тестирования новых функциональных возможностей системы. Первый из этих вариантов является более универсальным, однако, выполняется значительно дольше в случае необходимости выполнить тест для большого числа объектов. Заметим, что второй вариант требует внесения некоторых модификаций в механизмы тестирования Django, так как по умолчанию тестирование всегда проводится для создаваемой специально для него и изначально пустой базы данных. Для создания и удаления тестовой базы данных используется объекта класса django.test.runner.DiscoverRunner, и для того, чтобы проводить тестирование системы с использованием долговременной базы данных, необходимо создать класс-потомок этого класса с пустыми реализациями методов `setup_databases` и `teardown_databases`.

При использовании реляционной модели ЛРД вопрос о формировании списка свойств объектов, для которых необходимо выполнить тестирования, может являться достаточно сложным. Теоретически идеальным вариантом является для каждого из классов целевых объектов и каждой цепочки отношений выполнить тестирование для двух пар, состоящих из пользователя и объекта доступа. В первом случае пользователь должен быть связан с целевым объектом указанной

цепочкой отношений, а во втором случае пользователь и целевой объект не должны быть связаны. Выполнение полного набора тестов, однако, может быть затруднено двумя обстоятельствами. Первое из этих обстоятельств заключается в том, что при наличии в системе большого количества цепочек отношений тестирование может занимать весьма продолжительное время. Второе обстоятельство обусловлено тем, что для цепочек отношений, имеющих условия, обеспечение полноты тестового покрытия требует также проверки прав доступа пользователя к объекту, связанному с ним цепочкой отношений нужного вида, однако, для которой не выполнены соответствующие условия, зависящие от атрибутов входящих в цепочку объектов. Более подробно вопрос расчёта оптимального тестового покрытия рассматривается далее в подразделе 3.2 настоящего Приложения.

3.1.2 Использование встроенного тестового клиента Django

Вторым способом автоматизации тестирования приложений на основе Django является использование эмулятора Web-клиента. В этом режиме тестирующая подпрограмма формирует HTTP-запрос, после чего выполняются тестируемые функции, которые выполнялись бы при получении такого запроса от пользователя. Эти функции формируют HTTP-ответ, который анализируется тестирующей подпрограммой, которая проверяет соответствие вида передаваемой пользователю информации, ожидаемому. Данный метод тестирования является более высокоуровневым по сравнению с представленным в п. 3.1.1, так как тестируется вид страницы с точки зрения конечного пользователя и при этом игнорируются детали происходящего в процессе формирования страницы. Тестирование приложений данным способом необходимо в связи с тем обстоятельством, что Django позволяет использовать косвенно-вызываемые функции, выполняющие как предобработку входящих HTTP-запросов, так и постобработку ответов системы. Таким образом, разработчику может оказаться затруднительным видеть сразу все фрагменты кода системы, выполняемые в ответ на определенные запросы

пользователя. В случае, если формирование HTTP-ответа является сложной процедурой, данный метод тестирования обнаруживает ошибку, на какой бы из стадий формирования ответа она не произошла. В этом случае, однако, тестирование не дает информации о локализации ошибки, а только сообщает об ее наличии. С целью локализации ошибки впоследствии потребуется проверка работы каждой из вызываемых при формировании ответа системы функций.

3.2 Методы разработки наиболее полного тестового покрытия для реляционной модели логического разграничения доступа

Как отмечено в п. 3.1.1, при автоматизации тестирования механизмов реляционной модели логического разграничения доступа построение набора тестов, обеспечивающих проверку корректности обработки всех возможных цепочек отношений представляет собой нетривиальную задачу. Необходимо обеспечить тестирование механизмов разграничения доступа для пользователей и целевых объектов, связанных различными предусмотренными в системе цепочками отношений. Необходимо также обеспечить тестирование цепочек объектов, которые попарно связанны между собой примитивными отношениями, составляющими цепочку, для атрибутов которых не выполнено условие цепочки – например, удостовериться, что ответственный по подразделению не имеет право редактировать объекты, связанные с подразделением, если истек срок действия его полномочий.

Рассмотрим пример. Цепочка отношений, представленная на рисунке 1, включает четыре примитивных отношения и условие, связывающее атрибут целевого объекта с атрибутами одного из примитивных отношений, включенных в цепочку.

При этом одно из отношений – отношение вложенности объектов – является рефлексивным и транзитивным.

В учетом изложенного, для представленной на рисунке 1 цепочки необходимо проверить корректность предоставления пользователю права доступа к набору целевых объектов, содержащему не менее одного объекта, удовлетворяющего каждому из перечисленных далее условий.

- Статья, не имеющая автора, работающего в каком-либо подразделении, для которого пользователь является ответственным.
- Статья, имеющая автора, работающего в каком-либо подразделении, для которого пользователь непосредственно назначен ответственным.
- Статья, имеющая автора, работающего в каком-либо подразделении, дочернем для подразделения, для которого пользователь непосредственно назначен ответственным.
- Статья, имеющая автора, работающего хотя бы в одном подразделении, дочернем для одного из дочерних подразделений подразделения, для которого пользователь непосредственно назначен ответственным.
- Статья, имеющая автора, который работает или работал в подразделении, подконтрольном пользователю, но не работал в нем на момент написания статьи.

При такой проверке существует большое число способов увеличить детализацию тестового покрытия. Последнее условие может быть, например, комбинировано с тремя предыдущими, что даст в результате 7 пар отношений, включающих пользователя и целевой объект. Кроме того, условие попадания даты публикации статьи в период работы пользователя в подразделении может быть нарушено разными способами. Статья может быть опубликована как до поступления автора на работу в подразделение, подконтрольное пользователю, так и после его увольнения. Кроме того, автор статьи может работать в подразделении, подконтрольном пользователю, в настоящее время, однако также возможна и

ситуация, при которой сотрудник не работает в данном подразделении в настоящее время, но работал в нем на момент выхода статьи. Таким образом, попытка учесть все комбинации атрибутов пользователя и целевого объекта, влияющие на права доступа к этому объекту, приводит к недопустимо большому набору тестовых данных. По результатам анализа такого набора можно предложить следующие далее подходы к формированию тестовых наборов данных.

- Выделение групп пользователей, которые в большинстве случаев получают права доступа к объектам за счет цепочек отношений, имеющих некоторое количество одинаковых элементов в начале. К таким группам пользователей относятся, например, следующие группы: «суперпользователи», «ответственные по организациям или подразделениям», «ученые секретари диссертационных советов» или «руководителе НИР». При этом в набор данных для тестирования механизмов разграничения доступа обязательно должен быть включен хотя бы один пользователь, входящий в каждую из таких групп, и хотя бы один пользователь, не входящий в каждую из таких групп.

- Регулярное тестирование на максимально широком наборе данных через большие временные промежутки – например, перед выпуском очередной стабильной версии программной системы. При этом после внесения модификаций в политику безопасности системы тестирование проводится только для тех объектах, права доступа к которым с большой вероятностью могут измениться после внесения этих модификаций.

- Наиболее тщательное тестирование функций, которые можно считать с большей вероятностью содержащими ошибки. В [5] представлен ряд подходов к задаче оценки вероятности сбоев в том или ином модуле программной системы. В настоящее нельзя сделать вывод о применимости методов оценки количества дефектов программного обеспечения,

рассмотренных в [5], к исходному коду ИАС «ИСТИНА». Анализ применимости таких методов в ИАС «ИСТИНА» является одним их перспективных направлений дальнейших исследований.

Литература

[1] Реляционная модель логического разграничения доступа на основе цепочек отношений / В. А. Васенин, А. А. Иткес, К. А. Шапченко, В. Ю. Бухонов // Программная инженерия. — 2015. — № 9. — С. 11–19.

[2] Модели логического разграничения доступа в многопользовательских системах управления наукометрическим контентом / В. А. Васенин, А. А. Иткес, В. Ю. Бухонов, А. В. Галатенко // Программная инженерия. — 2016. — Т. 7, № 12. — С. 547–558.

[3] Владимир Дронов. Django: практика создания Web-сайтов на Python. СПб., БХВ-Петербург, 2016.

[4] Р. А. Сузи. Язык программирования Python. М., Интернет-Университет Информационных Технологий, БИНОМ, 2010

[5] Идеальная разработка ПО. Секреты лучших программистов. / Под ред. Энди Орама и Грега Уилсона. СПб., Питер, 2012.